Olika tillvägagångssätt för att balansera en mångfald strategier i ett spel

Kasper Renlund 1800126
Kandidatavhandling inom Datateknik
Handledare: Jan Westerholm
Fakulteten för Naturvetenskaper och Teknik
Åbo Akademi
2022

Innehåll

1 Introduktion **1**

2 Ett svårdefinierat begrepp[ändra rubrik] **3**

3 Handlingsberövning **5**

3.1 Handlingar 5

3.2 Metodik för handlingsberövning 5

4 Monte Carlo Tree Search **7**

5 Intuitiv balans genom design **11**

6 Kontinuerligt uppdaterade spel **13**

6.1 Power Creep 13

6.2 Metagame 13

7 Differentiering **15**

7.1 Material 15

7.2 Karaktärer/Enheter 15

7.3 Strategier 15

8 Case Study **17**

8.1 Bakgrund 17

8.1.1 Fiender 17

8.1.2 Torn 18

8.1.3 Rundor 18

8.2 Metod 19

8.3 Iterationer 19

8.4 Slutsatser 19

9 Diskussion **21**

Introduktion

Spelbalans är ett brett begrepp vars betydelse är olika beroende på vem man frågar. Balans sträcker sig från vad som gör ett spel roligt till vad som gör ett spel rättvist; ett spel kan vara rättvist utan att vara roligt, och roligt utan att vara rättvist. Förutom dessa bör spel även ha en viss svårighetsgrad, något som gör spelet givande att spela.

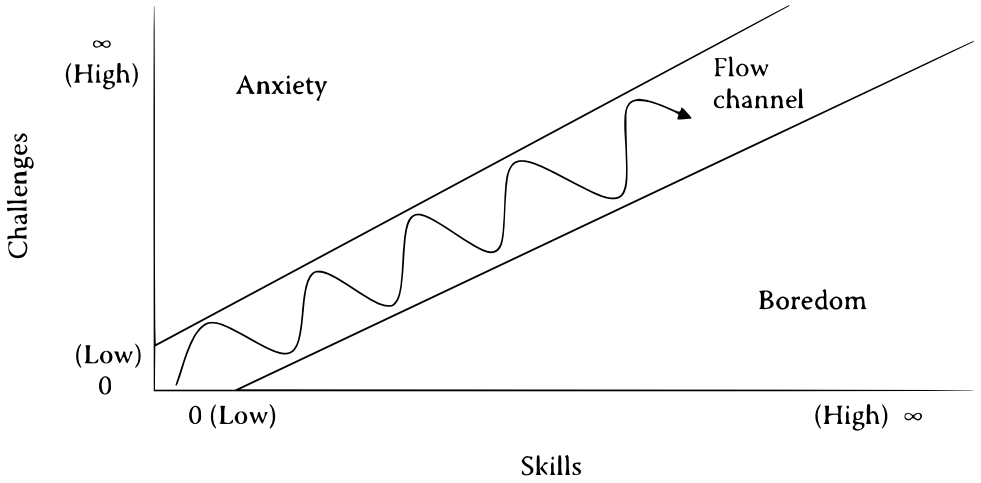
Med själva balanseringsprocessen syftar man ofta på det som sker efter att spelets grundfunktionalitet är färdig. Processen är tidskrävande; utvecklaren måste försäkra sig om att alla enhetsparametrar (d.v.s. de parametrar som definierar en enhet, dess hastighet, styrka o.s.v.) är optimala, något som tidigare gjordes för hand och på intuition men i nuläget körs på dator med hjälp av diverse algoritmer och en skräddarsydd heurestik. Utvecklaren bör även låta spelet testas av s.k. speltestare, människor vars uppgift är att spela spelet och ge feedback på huruvida det känns bra att spela, ifall det finns moment i spelet som inte ger något mervärde, moment som känns tröttsamma eller dylikt. Med hjälp av den data som samlas in kan utvecklaren balansera spelet enligt bästa förmåga, något som absolut inte är trivialt. Särskilt i flerspelarspel (eng. *multplayer games*) bör utvecklaren kontinuerligt uppdatera spelet eftersom det hör till människors natur att optimera strategier och hitta spelstilar som utvecklaren antingen underskattat eller inte ens vetat att var möjliga. Balans är alltså inte någon väldefinierad process som går att lösa. Det finns inget objektivt *rätt* sätt att balansera; största delen beror på utvecklarens vision.

Målet med avhandlingen är att bygga upp en intuitiv förståelse för spelbalans, problemen utvecklare ställs inför, problemlösningsprocessen och vanliga fallgropar. Avhandlingen kommer inte att fokusera mer än nödvändigt på tekniken och algoritmerna bakom balans.

Ett svårdefinierat begrepp[ändra rubrik]

Trots att det inte finns något objektivt rätt sätt att balansera spel är det värt att i någon mån redogöra för frågeställningar inom spelbalans som kommer att fungera som utgångspunkt för allt jag behandlar i avhandlingen. I detta kapitel presenteras problemställningar för designers, till stor del baserade på [2].

Jesse Schell ger en inblick i hurudan metodik man kan använda sig av under balansprocessen i sin bok ”The Art of Game Design” [2], specifikt kapitel 13. Det första han diskuterar är *rättvisa*. Det är viktigt för spel att ge spelare lika möjligheter att vinna genom val av resurser innan spelets början, t.ex. i Risk där varje spelare i ordning får placera ut enheter i de tillgängliga territorierna. Spel som Risk kallas asymmetriska spel; spelarna har olika resurser och utgångslägen. Symmetriska spel, som Schack eller Go hanterar balansen av utgångslägen genom att ge bägge spelarna samma resurser. Dessa spel är dock i allmänhet snäppet obalanserade, eftersom en av spelarna alltid har ett extra tempo, d.v.s. spelaren som börjar är alltid en tur före den andra. Förutom rättvist utgångsläge bör även de strategier som är tillgängliga för spelare under spelets gång vara rättvist distribuerade; olika utgångslägen får inte ge upphov till olika mängd praktiska strategier eller strategier som är orättvisa[3]. Ett exempel på ett spel vars utgångslägen inte är balanserade är Mai-Star, vars 6 olika geishor alla är ungefär lika kraftfulla förutom en, vars parametrar alla har maxvärdet 5(till skillnad från de andra geishorna som alla har högt värde på max en av tre parametrar) och som i sin tur har en enda strategi som skiljer sig från de andras och resulterar i en närmare 100% sannolikhet att vinna [4].



Figur 2.1: Balansgång mellan en spelares färdigheter och ett spels svårighetsgrad [2][5]

Vidare förklarar Schell även att en lämplig svårighetsgrad bör hittas för att förhöja spelupplevelsen. Detta är främst riktat mot enspelarspel, eftersom svårighetsgraden i flerspelarspel sätts av mänskliga motståndare. Ur figur 1 kan vi utläsa att utmaningarna bör vara proportionerliga mot spelarens färdigheter. Detta går för de flesta spel att åstadkomma bl.a. genom att gradvis höja på svårighetsgraden allteftersom spelaren tar sig vidare genom spelet eller genom att låta spelaren välja svårighetsgrad bland ett godtyckligt antal alternativ. Det sistnämnda är dock ofta problematiskt eftersom implementationen av olika svårighetsgrad ofta enbart är att artificiellt höja eller sänka på fienders styrka, vilket ofta direkt strider mot denna Flow ChannelSchell beskriver; spelet blir antingen frustrerande eller tråkigt om spelaren väljer någon annan svårighetsgrad än den som utvecklaren ämnat, inte för att det i sig är fel på metoden utan för att utvecklare sällan vill sätta extra arbete på olika svårighetsgrad i form av att skapa nya spelmoment eller färdigheter hos fiender.

Meaningful choices

Skill vs Chance

Head vs Hands

Simplicity/Complexity

Handlingsberövning

Handlingsberövning (eng. *Restricted Play*) är ett sätt att evaluera en spelares tillgängliga handlingar genom att beröva spelaren en viss handling (den handling som är av intresse) och observera resultat. Resultat kan i det här fallet vara vinster jämfört med förluster före och efter handlingen berövats.

3.1 Handlingar

En handling är, i det här sammanhanget, vad som helst en utvecklare kan beröva spelaren. Det kan vara allt från att beröva spelaren möjligheten att använda en spelpjäs, till att beröva spelaren möjligheten att se på djup eller att reagera på motspelares handlingar. Man kan också tvinga en spelare att utföra en viss handling ett visst antal gånger, eller hindra spelaren från att utföra en viss handling fler än ett visst antal gånger.

3.2 Metodik för handlingsberövning

Som del av sin doktorsavhandling skapade Alexander Jaffe m.fl. [6] en grund för att med hjälp av så kallade AI agenter, alltså icke-mänskliga spelare, algoritmiskt evaluera olika handlingars inverkan på ett spel. I det här avsnittet går jag igenom deras resultat[hm] och utforskar sedermera hur man kan använda denna metod mera intuitivt, bl. a. i form av tankeexperiment, något som förvisso kräver en viss intim kunskap om spelet man evaluerar och antagligen lämpar sig mera för relativt simpla spel eller isolerade delar av ett mera komplext spel.

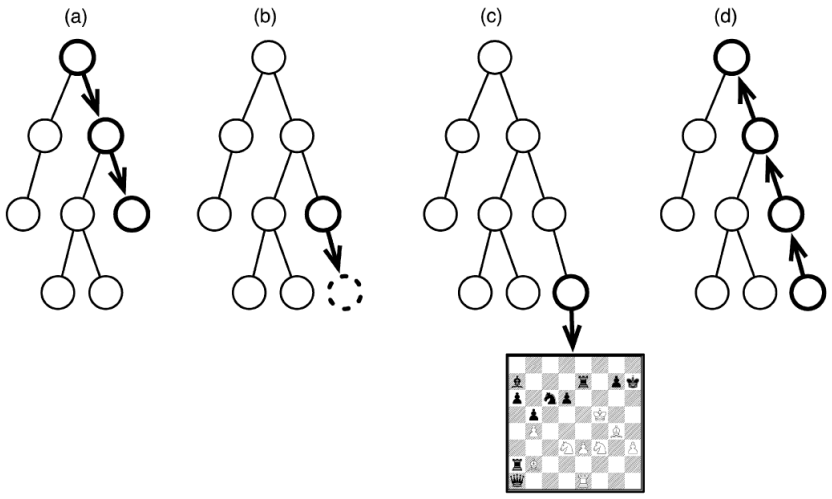
Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) är en sökalgoritm som lämpar sig bäst för spel med fullständig information (men har även framgångsrikt tillämpats i spel utan perfekt information [7]) vilket i allmänhet innebär bredspel eller diskreta datorspel, d.v.s. spel som inte spelas i realtid utan varje spelare har en tur under vilken de kan utföra vissa handlingar varefter turen går till nästa spelare.

Konceptet bakom MCTS är, liksom för alla Monte Carlo metoder, att ett datorprogram spelar en stor mängd slumpmässiga, simulerade spel och väljer följande drag enligt det som synligen leder till högsta vinstsannolikhet. Tanken bakom MCTS är att komma fram till ett svar betydligt snabbare än ren Monte Carlo vilket utförs genom att leda simuleringarna med ett evigt växande s.k. spelträd (eng. *Game Tree*). I teorin bör detta resultera i att de mera lovande noderna nås först och besöks oftare än mindre attraktiva noder [8]. MCTS är oftare tidsbegränsad än iterationsbegränsad, d.v.s. utvecklare specificerar en viss tid som de låter programmet köra innan det terminerar.

MCTS är en fyrstegsalgoritm där de fyra stegen är som följer [8] [4] [9]:

- Val - Programmet börjar från rotnoden `R`, (`R` representerar det nuvarande speltillståndet) och går från nod till nod tills en lövnod har nåtts. För lövnoden gäller att ett spel inte ännu simulerats från den.
- Expansion - Ifall den nyligen nådda lövnoden inte avslutar spelet direkt, genom vinst/förlust/oavgjort, så skapar programmet en eller flera nya barnnoder `B`. Dessa barnnoder representerar lagliga drag för programmet i det nuvarande speltillståndet. Programmet väljer en av dessa nya noder `B` genom att maximera en viss parameter, vilket diskuteras senare i kapitlet.
- Simulering - Fullgör (åtminstone) en slumpmässig spelning börjandes från den nyligen valda noden `B` till och med det att spelet tar slut.
- Tillbakaförökning (eng. Backpropagation) - Använd resultatet från föregående steg för att uppdatera de tidigare nådda nodernas data från nod `B` till nod `R`



Figur 4.1: MCTS fyra faser: val, expansion, simulering, tillbakaförökning [8]

Eftersom det i MCTS inte definieras hur man bör gå tillväga för att utföra varje steg bör MCTS ses på mera som en metod än en algoritm [8]. Man använder därmed olika, mer specifika, algoritmer som hjälpmedel.

Val av nod från rotnod `R` är en process som är nära släkt med den k-armade banditen, som diskuteras senare i kapitlet, eftersom programmet bör hitta en gyllene medelväg mellan att utforska möjligast många barnnoder (*exploration*) och att utforska lovande barnnoder (*exploitation*). Ett vanligt tillvägagångssätt för val av handling i MCTS är UCT (Upper Confidence Bound for Trees) [8] [10] :

$$U_i = \overline{X_i} + C\sqrt{\frac{2\ln N}{n_i}} \tag{4.1}$$

där C är en konstant som bestämmer ifall algoritmen är inriktad på att utforska (C är liten) eller exploatera (C är stor), N och n_i är antalet gånger föräldranoden till nod i respektive nod i besökts och $\overline{X_i}$ är ett enkelt medelvärde på en av utvecklaren förbestämd evalueringsfunktion för alla simuleringar som besökt noden i. Det finns variationer på (5.1) men de är alla mer eller mindre lika. I nollsummespel kan evalueringsfunktionen se ut på följande vis [11]:

$$\begin{aligned} eval_j = &\left(assetsNow_j^{self} - assetsBefore_j^{self}\right) \\ &- \left(assetsNow_j^{enemy} - assetsBefore_j^{enemy}\right) \end{aligned} \tag{4.2}$$

där $eval_j$ är differensen mellan egna och fiendens tillgångar före och efter simulation j [11].

Expansionsmetoden varierar betydligt mera från spel till spel än val av handling. Den beror på spelets tillståndsrymd (eng. *state space*), d.v.s. alla möjliga tillstånd spelet kan finna sig i, och dess förgreningsfaktor, d.v.s. hur många lagliga drag en spelare i snitt har under varje tur [8]. I allmänhet låter man dock expandera en nod ifall den besökts tillräckligt många gånger [8] [11] [4].

Även simulering är väldigt spelberoende men har i breda drag formen: utför drag från nyligen tillsatt barnnod `B` till och med det att spelet termineras. Beräkna belöningen för sekvensen enligt evalueringsfunktionen, i vårt fall (5.2). Tillbakaförökningen uppdaterar helt enkelt noderna enligt (5.1) och alla dessa fyra steg upprepas tills en tid T_{max} nåtts.

Intuitiv balans genom design

Kontinuerligt uppdaterade spel

6.1 Power Creep

6.2 Metagame

Differentiering

7.1 Material

7.2 Karaktärer/Enheter

7.3 Strategier

Case Study

8.1 Bakgrund

Spelet vi evaluerar är av typen Tower Defense (TD, sv. *tornförsvar*) och är ett hobbyprojekt skrivet av mig. Spelet är en grov kopia av den populära spelserien *Bloons Tower Defense* och går ut på att försvara en godtyckligt lång stig mot allt svårare vågor av fiender. Ifall en fiende når stigens slut dras dess kvarvarande skadepoäng(eng. *hit points*, även kallat hälsopoäng för att behålla förkortningen) från spelarens. Då spelarens skadepoäng når 0 är spelet över. Spelarens uppgift är att strategiskt placera ut olika torn som antingen kan skada eller uppehålla fienderna[bild].

8.1.1 Fiender

Spelets fiender är olika färgade s.k. ballonger (därav orginalspelets namn, *Bloons*) vilka är designade som Matrjosjkadockor(fig. 8.1).

Skadetyper Spelet har fyra huvudsakliga skadetyper: `Sharp`, som kan skada alla ballonger förutom bly; `Magic`, som kan skada alla ballonger förutom bly och lila; `Fire`, som kan skada alla ballonger förutom lila och slutligen `Explosive`, som kan skada alla ballonger förutom svarta.
Spelet har även kamoflageballonger, vilka kan vara av bastyperna röd till och med pink. Kamoflageballonger kan endast ta skada av torn som har möjlighet att se genom kamoflagen.

Namn	Anteckningar	Hastighet	Skadepoäng(tot)	Förälder till	Barn till
Red		1,00	1	None	Blue
Blue		1,40	2	Red Bloon	Green
Green		1,80	3	Blue Bloon	Yellow
Yellow		3,20	4	Green Bloon	Pink
Pink		3,50	5	Yellow Bloon	Black, White
Black	Immun mot explosioner	1,80	11	Pink Bloon x2	Zebra
White	Immun mot nedfrysning(finns inte i spelet ännu)	2,00	11	Pink Bloon x2	Zebra
Purple	Immun mot magi och eld	3,00	11	Pink Bloon x2	None
Lead	Immun mot skarpa projektiler	1,00	23	Black Bloon x2	None
Zebra	Immun mot explosioner och nedfrysning	1,80	23	Black Bloon White Bloon	Rainbow
Rainbow		2,20	47	Zebra Bloon x2	Ceramic
Ceramic	Tar 10 träffar att förgöra	2,50	104	Rainbow Bloon x2	Boss
Boss	Tar 200 träffar att förgöra	1,00	616	Ceramic x4	

Figur 8.1: temp

8.1.2 Torn

I spelet finns fyra spelbara torn:

Albin - Ett generiskt torn som kastar projektiler i en rak linje. Kan se kamoflageballonger ifall spelaren investerat i flera uppgraderingar. Medellång räckvidd.

Tack Shooter - Ett torn med kort räckvidd som avfyrar 8-16 projektiler runt sig.

Cannon - Ett torn som skjuter bomber med liten explosionsradie. Lång räckvidd.

Magickan - Ett torn som kastar magiska projektiler på fiender. Medellång räckvidd.

Uppgraderingar Varje torn har två uppgraderingsstigar(eng. *upgrade path*). Vissa uppgraderingar ger torn ökad skada medan andra ger dem större användbarhet (eller verktyg, eng. *utility*): större räckvidd, lägre intervall mellan attacker, nya projektiler eller tillgång till kamoflageballonger.

8.1.3 Rundor

Spelet har för tillfället 40 spelbara rundor, där rundorna blir progressivt svårare och kulminerar i en s.k. *Boss Fight*, d.v.s. en fiende som är starkare än andra fiender.
En runda består av ett godtyckligt antal vågor. En våg kan innehålla ett godtyckligt antal ballonger, även flera olika sorters ballonger. Varje våg har en variabel `timeUntilNext` som beskriver intervallet mellan nuvarande och nästa våg. För att vidare kunna justera en rundas tempo har varje våg även en variabel `interval` som beskriver intervallet mellan ballonger (se *spawn rate*). Varje runda måste ta slut innan nästa runda kan påbörjas.

8.2 Metod

Restricted play för spelaren
Evaluera fiender och rundor me nå matematikfundering Försök matcha rundornas ss-calingtill en viss kurva m.h.a. chi-squared.

8.3 Iterationer

8.4 Slutsatser

Diskussion

Referenser

[1] A. Jaffe, ”Understanding Game Balance With Quantitative Methods,” diss., University of Washington, 2013, kap. Abstract.

[2] J. Schell, *The Art of Game Design*. CRC Press, 2015.

[3] D. Sirlin, ”Balancing Multiplayer Competitive Games,” *Game Developer's Conference*, 2009.

[4] E. Klementev, A. Fedorovskaya, F. Hakimov, H. Aslam och J. A. Brown, ”Monte Carlo Tree Search player for Mai- Star and Balance Evaluation,” i *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, s. 686–691.

[5] G. Chanel, C. Rebetez, M. Bétrancourt och T. Pun, ”Boredom, Engagement and Anxiety as Indicators for Adaptation to Difficulty in Games,” ser. MindTrek '08, Tampere, Finland: Association for Computing Machinery, 2008, 13–17, ISBN: 9781605581972. URL: https://doi.org/10.1145/1457199.1457203.

[6] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin och Z. Popović, ”Evaluating Competitive Game Balance with Restricted Play,” 2012.

[7] J. Wang, T. Zhu, H. Li, C.-H. Hsueh och I.-C. Wu, ”Belief-state Monte-Carlo tree search for Phantom games,” i *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, s. 267–274.

[8] P. Ciancarini och G. P. Favini, ”Monte Carlo tree search in Kriegspiel,” *Artificial Intelligence*, årg. 174, nr 11, s. 670–684, 2010, ISSN: 0004-3702. URL: https://www.sciencedirect.com/science/article/pii/S0004370210000536.

[9] P. I. Cowling, E. J. Powley och D. Whitehouse, ”Information Set Monte Carlo Tree Search,” *IEEE Transactions on Computational Intelligence and AI in Games*, årg. 4, nr 2, s. 120–143, 2012.

[10] L. Kocsis och C. Szepesvári, ”Bandit Based Monte-Carlo Planning,” i *Machine Learning: ECML 2006*, J. Fürnkranz, T. Scheffer och M. Spiliopoulou, utg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 282–293.

[11] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada och R. Thawonmas, ”Application of Monte-Carlo tree search in a fighting game AI,” i *2016 IEEE 5th Global Conference on Consumer Electronics*, 2016, s. 1–2.

[12] ”The Art of Game Design,” i CRC Press, 2015, kap. 13.